

# PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-185245

(43)Date of publication of application : 16.07.1996

(51)Int.Cl.

G06F 1/24  
G06F 11/28

(21)Application number : 06-339959

(71)Applicant : HEWLETT PACKARD JAPAN LTD  
TOSHIBA CORP

(22)Date of filing : 28.12.1994

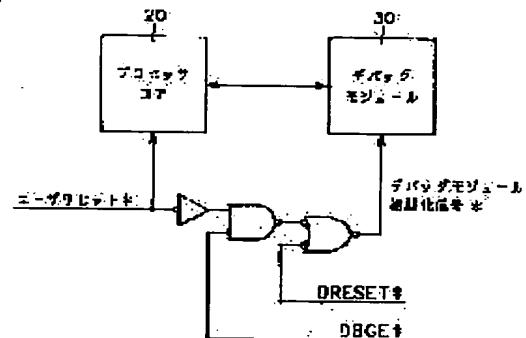
(72)Inventor : YANO TATSUO  
MIYAMORI TAKASHI

## (54) METHOD FOR RESETTING MICROPROCESSOR

### (57)Abstract:

**PURPOSE:** To observe the behavior right after the resetting of a microprocessor including a circuit for providing part of a debugging function from a debugging tool by being supported by the circuit.

**CONSTITUTION:** When a signal DBGE\* showing that the debugging tool is connected to the microprocessor is active (at low level), a processor core 20 is initialized even when a user reset signal is supplied, but the reset signal does not reach a debugging module 30. Consequently, the debugging module 30 maintains its function even right after user resetting, so abnormal operation of a target system, etc., in the above state can be acquired by the debugging tool.



(19)日本国特許庁 (JP)

## (12) 公開特許公報 (A)

(11)特許出願公開番号

特開平8-185245

(43)公開日 平成8年(1996)7月16日

(51) Int.Cl.<sup>8</sup>  
G 0 6 F 1/24  
11/28

識別記号 庁内整理番号  
L 7313-5B

F I

技術表示箇所

G 0 6 F 1/ 00 3 5 0 B

審査請求 有 請求項の数 2 FD (全 23 頁)

(21)出願番号 特願平6-339959  
(22)出願日 平成6年(1994)12月28日

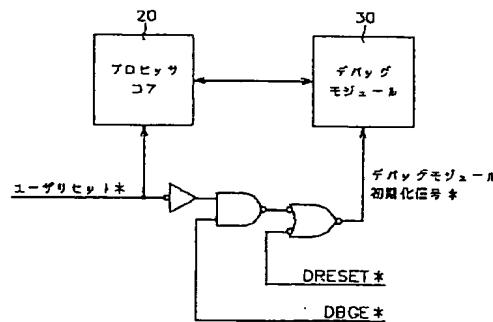
(71)出願人 000121914  
日本ヒューレット・パッカード株式会社  
東京都八王子市高倉町9番1号  
(71)出願人 000003078  
株式会社東芝  
神奈川県川崎市幸区堀川町72番地  
(72)発明者 矢野 達男  
東京都八王子市高倉町9番1号 横河・ヒューレット・パッカード株式会社内  
(72)発明者 宮森 高  
神奈川県川崎市幸区堀川町580番1号 株式会社東芝半導体システム技術センター内  
(74)代理人 弁理士 上野 英夫

(54)【発明の名称】マイクロプロセッサのリセット方法

## (57)【要約】

【目的】デバッグ機能の一部を実現する回路を含むマイクロプロセッサに対してリセットがかかった直後の挙動を、デバッギングツールからこの回路の支援を得て観測できるようにする。

【構成】マイクロプロセッサにデバッギングツールが接続中であることを示す信号DBG#\*がアクティブ(ローレベル)の場合には、ユーザリセット信号が与えられてもプロセッサコア20は初期化されるが、デバッグモジュール30にはリセット信号が到達しない。これにより、ユーザリセット直後であってもデバッグモジュール30は機能を維持しているので、この状態でのターゲットシステムの異常動作などをデバッギングツールで捕捉することができる。



## 【特許請求の範囲】

【請求項1】マイクロプロセッサの外部に置かれるデバッグツールと接続してデバッグ機能を実現する論理手段を内蔵するマイクロプロセッサのリセット方法において、デバッグ機能を実現するための前記マイクロプロセッサ内のロジック回路のユーザリセット信号による初期化は、前記デバッグツールが前記マイクロプロセッサに接続されていない場合にのみ行われることを特徴とするマイクロプロセッサのリセット方法。

【請求項2】前記デバッグツールが前記マイクロプロセッサへ接続されているか否かの判断は、前記デバッグツールが前記マイクロプロセッサに接続された際に前記デバッグツールから供給される特定の信号の有無を検出することによって行うことを特徴とする請求項1記載のマイクロプロセッサのリセット方法。

## 【発明の詳細な説明】

## 【0001】

【産業上の利用分野】本発明は、マイクロコンピュータを応用したシステムのデバッグを行なうデバッグシステムに関する。

## 【0002】

【従来技術】図1に従来例1を示す。従来例1は、一般にROMモニタと呼ばれるデバッグシステムである。ユーザターゲットシステム70上に、ホストコンピュータ90と接続するためのシリアルインターフェース80を用意し、メモリ40中に、モニタプログラム41を格納しておく。マイクロプロセッサ10は、このモニタプログラム41を走らせることにより、I/O50、メモリ40、レジスタ11にアクセスする。さらにソフトウェアブレイク命令を使用して、ユーザプログラムの実行制御を行なう。

【0003】図2に従来例2を示す。ユーザターゲットシステム70上のマイクロプロセッサ10中に、デバッグツール100との通信に必要なシリアルインターフェース12と、デバッグツール100から送られてきた電気信号を解釈して実行するシーケンサ13を内蔵する。シーケンサ13は、送られてきた信号に従って、ユーザプログラムの実行を一時停止し、レジスタ11にアクセスしたり、バスコントローラ14を使用して、メモリ40やI/O50にアクセスする。さらに、ハードウェアブレークポイントや、ソフトウェアブレーク命令を使用して、ユーザプログラムの実行制御を行う。シリアルインターフェース12からの信号は、直接には、ホストコンピュータ90に接続できないことが多いので、デバッグツール100が、ホストコンピュータ90からのコマンドをマイクロプロセッサ10に接続できる電気信号に変換したり、またマイクロプロセッサ10からの信号をホストコンピュータ90が理解できるデータ形式に変換する。

【0004】図3に従来例3を示す。従来例3は一般にインサーキットエミュレータと呼ばれるデバッグシステムである。デバッグ時には、ユーザターゲットボードシステム70上のマイクロプロセッサ10を取り外すか、または無効にし、この部分にデバッグツール110のプロープを接続して、デバッグ用マイクロプロセッサ120を代わりに動作させる。デバッグ用マイクロプロセッサ120は、デバッグツール上のモニタプログラムメモリ130に格納されているモニタプログラムを実行することにより、ユーザプログラムの実行を制御したり、メモリ40上のデータにアクセスしたり、I/O50をアクセスする。また、デバッグ用マイクロプロセッサ120は、あたかもマイクロプロセッサ10が実行しているかのように、ユーザターゲットシステム上のメモリ40に格納されているプログラムを実行する。また、デバッグツール110は、トレースメモリ140を有し、デバッグ用マイクロプロセッサ120のプロセッサバスの状態をトレースすることができる。デバッグ用マイクロプロセッサ110は、マイクロプロセッサ10からは得られないトレース情報を出力する。これにより、プロセッサバスからだけではトレースできないプロセッサの内部状態の一部がトレース可能になる。

【0005】図4に従来例4を示す。従来例4は、一般にプリプロセッサと呼ばれているデバッグシステムである。ユーザターゲットシステム70上のマイクロプロセッサ10のプロセッサバス90に、ロジックアナライザ150のプロープを接続することにより、マイクロプロセッサ10のメモリ40やI/O50に対するアクセスをトレースすることができる。

【0006】なお、図1ないし図4を用いて説明した従来例の具体的な動作や回路構成は当業者には周知であるため、これ以上詳細な説明はここでは与えない。

## 【0007】

【発明が解決しようとする課題】従来例1では、モニタプログラムをユーザメモリ上で走らせるため、ユーザのユーザターゲットシステムのメモリシステムの動作が完全でない場合にモニタ自身が安定して動作しない場合があった。また、ターゲットシステムのメモリ量に余裕がない場合、モニタが占有するアドレス空間が確保できないことがあった。さらに、モニタモードへ入るためにユーザの割込みの一部を使用しなければならないので、プログラムの種類によってはデバッグが不可能なことがあった。また、あらかじめユーザターゲット上にシリアルインターフェースの回路を実装しなければならないなど、デバッグ後は使用しないかもしれない回路をターゲットシステムに設ける必要があった。また、デバッグのためのハードウェアブレークポイント等の資源をもつてないので、デバッグ機能が貧弱であり、トレースも取れなかった。

【0008】従来例2では、シーケンサをマイクロプロ

セッサに内蔵し、さらに、シーケンサがレジスタにアクセスするため、デバッグツールとの接続のためのロジック回路が複雑になり、チップ上の面積も大きくなつた。また、レジスタの追加などが発生した場合、シーケンサを変更しなければならないので、変更のための作業が大きかつた。さらに、この従来例でもトレースは取れなかつた。

【0009】従来例3では、ユーザターゲット上のマイクロプロセッサのすべてのピンにデバッグツールを接続するため、プローブが高価になり、しかもプローブの接触が不安定になることが多かつた。また、ターゲット上のメモリにアクセスする場合とデバッグツール内のモニタメモリにアクセスする場合とで、高速にバスを切替えなければならないため、動作周波数の高いプロセッサでは実現が困難であった。派生品のマイクロプロセッサがある場合には、パッケージやピンの数、位置が異なるために、本質的には同じデバッグツールが使用できるにも関わらず、プローブを派生品ひとつひとつに対して別のデバッグツールを用意しなければならなかつた。また、プローブを接続することにより、ユーザターゲットで使用する信号にも影響を与え、ユーザターゲットの動作そのものを不安定にする場合があつた。

【0010】従来例4では、トレースに関しては、高速のプロセッサに対しても有効であるが、内蔵キヤッシュメモリを持つプロセッサでは、キヤッシュにヒットしている場合のトレースは取れなかつた。また、内部にキューを持つプロセッサの場合は、フェッチされた命令が実行されたかどうかを判別できなかつた。また、ユーザプログラムの実行制御機能はなく、ユーザメモリやI/Oの内容を参照したりすることはできなかつた。

#### 【0011】

【課題を解決するための手段】従来例1の問題を解決するために、本発明では、モニタメモリをデバッグツール内に格納し、デバッグツール専用の信号を使ってモニタプログラムを動作させ、デバッグツール専用の割込みを用意し、ハードウェアブレーク機能をマイクロプロセッサに内蔵した。

【0012】従来例2の問題を解決するために、本発明では、モニタプログラムを使用してレジスタ、ユーザメモリ、I/Oにアクセスする。

【0013】従来例3の問題を解決するために、本発明では、デバッグツール内のモニタメモリでモニタプログラムを走らせるためのロジック回路、ハードウェアブレークポイントを含む実行制御を行なうためのロジック回路、及び実行された命令のPC情報を出力するためのロジック回路をマイクロプロセッサに内蔵した。また、デバッグツールとの接続専用のピンをマイクロプロセッサに用意し、これらのピンだけをデバッグツールと接続した。さらにモニタプログラム実行中だけ出力されるクロックの周波数を下げる機能を付け加えた。

【0014】従来例4の問題を解決するために、本発明では、デバッグツール内のモニタメモリでモニタプログラムを走らせるためのロジック回路、ハードウェアブレークポイントを含む実行制御を行なうためのロジック回路、及び実行された命令のPC情報を出力するためのロジック回路をマイクロプロセッサに内蔵した。

#### 【0015】

【実施例】以下に図面を用いて本発明をアドレス32ビット、データバス32ビットのプロセッサコアを内蔵するマイクロプロセッサに適用した場合の実施例を示す。なお、図面及び以下の説明中、信号名に付加されたアスターisk(\*)は、その信号が負論理であることを表す。

【0016】<デバッグシステム全体の説明>図5に本発明の一実施例のデバッグシステムの構成図を示す。デバッグシステムは、ユーザターゲットシステム70とデバッグツール60からなる。ユーザターゲットシステム70は、デバッグ機能を内蔵したマイクロプロセッサ10、メモリ40、I/O50から構成される。マイクロプロセッサ10は、プロセッサコア20とデバッグモジュール30から構成される。プロセッサコア20は、プロセッサバス20を介して、メモリ40やI/O50をアクセスし、プログラムを実行する。また、プロセッサコア20は、内部デバッグインターフェース15と内部プロセッサバス16によって、デバッグモジュール30と接続されている。デバッグモジュール30は、外部デバッグインターフェース71によって、デバッグツール60と接続されている。

【0017】<実行モードの説明>デバッグシステムには、モニタプログラムを実行しているデバッグモードと、ユーザプログラムを実行しているノーマルモードの2つの実行モードがある。

【0018】[デバッグモードの説明] プロセッサコア20において、デバッグ例外、あるいはデバグリセットが発生すると、デバッグ例外のベクタアドレス、あるいはデバグリセットのベクタアドレスへジャンプし、デバッグモードに入る。これらのベクタアドレスに対応するメモリは、デバッグツール60上に置かれている。プロセッサコア20は、デバッグモジュール30を介して、デバッグツール60上のモニタプログラムを実行する。モニタプログラムでは、メモリ、I/O、及びレジスタのリードやライト、ハードウェアブレークポイントの設定、ユーザプログラムの実行開始アドレスの指定などの実行制御機能を実現する。プロセッサコア20がノーマルモードへの復帰命令を実行することによって、ノーマルモードへ復帰し、ユーザプログラムの実行を開始あるいは再開する。

【0019】[ノーマルモードの説明] ノーマルモードでは、デバッグシステムは、ユーザプログラムを実行する。ノーマルモードでは、外部デバッグインターフェース71にPC(program counter)情報が出力される。デ

バグシステムは、ハードウェアブレーク、ソフトウェアブレーク、デバッグ割込み、デバッグリセットなどによって、プロセッサコア20へデバッグ例外またはデバッグリセットを要求し、デバッグモードへ制御を移す。

【0020】<デバッグモジュールの機能の概略説明>以下に、デバッグモードで有効なシリアルモニタバス機能と、ノーマルモードで有効な、PCトレース機能、トレーストリガ機能、ハードウェアブレーク機能、ソフトウェアブレーク機能、デバッグ割込み機能、デバッグリセット機能、及びマスク機能について、説明する。

【0021】【シリアルモニタバス機能の説明】シリアルモニタバス機能は、プロセッサコア20がモニタ専用のアドレス領域をアクセスしている場合に、デバッグ専用のピンを通るシリアル伝送経路を通じてデバッグツール60内のモニタメモリにアクセスすることによって実現される。モニタ専用メモリ以外の領域をアクセスする場合には、通常のプロセッサバスを通じたアクセスが行なわれる。これにより、モニタは、ユーザターゲットシステム上のメモリやI/Oもアクセスできる。なお、以下の実施例ではシリアルモニタバスのビット幅は1ビットであるが、マイクロプロセッサのピンをこのバスのためにもっと多く使用できる場合には、複数ビット幅にすることもできる。

【0022】【PCトレース機能の説明】PCトレース機能は、プロセッサコア20がユーザプログラムを実行している時に、そのプログラムカウントPCの値をトレースする機能である。プロセッサコア20がメモリ40上のユーザプログラムを実行しているときに、そのPCトレース情報を内部デバッグインターフェース15へ出力し、デバッグモジュール30がその情報を入力して情報を加工した後、外部デバッグインターフェース71を介して、デバッグツール60へ出力することによって実現される。

【0023】【トレーストリガ機能の説明】トレーストリガ機能には、命令アドレストレーストリガ、データアドレスブレーク、及びプロセッサバストレーストリガの3種類がある。命令アドレストレーストリガ機能は、プロセッサコア20が内部デバッグインターフェースに出力する命令実行アドレスの値を、デバッグモジュール30内のレジスタに設定されているアドレスの値と比較し、それらが一致した時に、トレーストリガが発生したことを外部デバッグインターフェース71を介して、デバッグツール60へ伝達することによって、実現される。データアドレストレーストリガ機能は、プロセッサコア20が内部デバッグインターフェースに出力するデータアクセスアドレスの値を、デバッグモジュール30内のレジスタに設定されているアドレスの値と比較し、それらが一致した時に、トレーストリガが発生したことを外部デバッグインターフェース71を介してデバッグツール60へ伝達することによって、実現される。プロ

セッサバストレーストリガ機能は、プロセッサコア20がプロセッサバスに出力するデータアクセスアドレス及びそのデータの値を、デバッグモジュール30内のレジスタに設定されているアドレス及びデータの値と比較し、それらが一致した時に、トレーストリガが発生したことを外部デバッグインターフェース71を介してデバッグツール60へ伝達することによって、実現される。

【0024】【ハードウェアブレーク機能の説明】ハードウェアブレーク機能には、命令アドレスブレーク、データアドレスブレーク、及びプロセッサバスブレークの3種類がある。命令アドレスブレーク機能は、プロセッサコア20が内部デバッグインターフェースに出力する命令実行アドレスの値を、デバッグモジュール30内のレジスタに設定されているアドレスの値と比較し、それらが一致した時に、デバッグ例外をプロセッサコア20へ要求することによって、実現される。データアドレスブレーク機能は、プロセッサコア20が内部デバッグインターフェースに出力するデータアクセスアドレスの値を、デバッグモジュール30内のレジスタに設定されているアドレスの値と比較し、それらが一致した時に、デバッグ例外をプロセッサコア20へ要求することによって、実現される。プロセッサバスブレーク機能は、プロセッサコア20がプロセッサバスに出力するデータアクセスアドレス及びそのデータの値を、デバッグモジュール30内のレジスタに設定されているアドレス及びデータの値と比較し、それらが一致した時に、デバッグ割込みをプロセッサコア20へ要求することによって、実現される。

【0025】【ソフトウェアブレーク機能の説明】ソフトウェアブレーク機能は、プロセッサコア20がソフトウェアブレーク命令を実行することによりデバッグ例外を発生するものである。これにより、デバッグモードへ移行する。

【0026】【デバッグ割込み機能の説明】デバッグ割込み機能は、プロセッサ20のデバッグ割込み信号をアサートすることにより、プロセッサ20がデバッグ例外を発生する機能である。これにより、デバッグモードへ移行する。

【0027】【デバッグリセット機能の説明】デバッグリセット機能は、プロセッサ20のデバッグリセット信号をアサートすることにより、プロセッサ20がデバッグリセットを発生する機能である。これにより、プロセッサ20とデバッグモジュール30の内部状態が初期化され、プロセッサコア20はデバッグモードとなり、デバッグリセットのペクタアドレスからプログラムの実行を開始する。

【0028】【マスク機能の説明】マスク機能は、設定に従って、ノーマルモード中のユーザ割込みをマスクしたり、デバッグモード中のユーザリセットをマスクしたりする。

【0029】<デバッグモジュールの全体構成の詳細説明>次にデバッグモジュール30について、さらに詳しく説明する。

【0030】図6にデバッグモジュール30の内部プロックを示す。デバッグモジュール30には、命令/データアドレスブレーク回路31、PCトレース回路32、プロセッサバスブレーク回路33、シリアルモニタバス回路34、レジスタ回路35、外部インターフェース回路36、分周回路37が設けられている。

【0031】PCトレース回路32は、プロセッサコア20と内部デバッグインターフェース15で接続されている。プロセッサ20から出力される、実行された命令のPC情報を入力し、その情報を加工して、外部インターフェース回路36へ出力する。

【0032】命令/データアドレスブレーク回路31は、プロセッサコア20と内部デバッグインターフェース15で接続されている。プロセッサコア20から出力される命令アドレスを入力し、その値がレジスタ回路35内に設定されている命令アドレスと一致した場合、命令アドレスブレークの使用が設定されていれば、プロセッサコア20へ命令アドレスブレーク例外を要求する。このとき、トレーストリガの使用が設定されていれば、トリガが発生したことをPCトレース回路32へ通知する。

【0033】また、プロセッサコア20から出力されるデータアドレスを入力し、その値がレジスタ回路35内に設定されているデータアドレスと一致した場合、データアドレスブレークの使用が設定されていれば、プロセッサコア20へデータアドレスブレーク例外を要求する。このとき、トレーストリガの使用が設定されていれば、トリガが発生したことをPCトレース回路32へ通知する。ブレークまたはトレーストリガの使用は、レジスタ回路35内の当該レジスタの当該ビットの値によって設定される。

【0034】プロセッサバスブレーク回路33は、内部プロセッサバス16を介して、プロセッサコア20と接続されている。データは、ビット毎にマスクをすることができる。この回路は、プロセッサバス上のバスサイクルを監視し、レジスタ回路35に設定されたアドレス及びデータと、発生したバスサイクルのアドレス及びデータが一致すると、プロセッサコア20へ例外を要求する。このとき、トレーストリガの使用が設定されていれば、トリガが発生したことをPCトレース回路32へ通知する。

【0035】シリアルモニタバス回路34は、内部プロセッサバス16を介してプロセッサコア20と接続され、プロセッサコア20がデバッグツール60上のモニタプログラムを実行するときに、並列形式のデータを直列形式に変換し、または直列形式のデータを並列形式に変換し、そのインターフェースを行なう。

【0036】レジスタ回路35は、デバッグモジュールの機能を制御する制御レジスタを含む。各レジスタ毎にアドレスが割り当てられている。内部プロセッサバス16と内部デバッグインターフェース15によってプロセッサコア20と接続され、モニタプログラムを走らせるこことによって制御レジスタの内容をリードまたはライトすることができる。また、制御レジスタの内容は、デバッグモジュール30内の各回路やプロセッサコア20へ出力され、デバッグ機能が制御される。

【0037】外部インターフェース回路36は、デバッグモジュール30内のPCトレース回路32、シリアルモニタバス回路34、プロセッサコア20とデバッグツール60とのインターフェースを制御する回路である。また、マスク機能も外部インターフェース回路36内で実現されている。

【0038】分周回路37は、クロック信号CLKを分周する回路である。シリアルモニタバス回路は、分周したクロックCLK2で動作する。

【0039】<デバッグツールの全体構成の詳細説明>

【0039】図7にデバッグツールの全体構成を示す。デバッグツール60には、通信インターフェース620、コントローラ630、モニタメモリ640、モニタメモリインターフェース650、トレースメモリインターフェース660、トレースメモリ670、ランコントロール680、及びターゲットインターフェース690が設けられている。

【0040】通信インターフェース620はホストコンピュータとの通信を行なう。コントローラ630は、通信インターフェース620を経由してホストコンピュータから送られてきたコマンドを解析し、実行し、結果を送り返す。モニタメモリ640は、モニタプログラムを格納し、これを実行するためのメモリである。モニタメモリインターフェース650は、ユーザターゲットシステム70からの直列形式の信号をモニタメモリ640にアクセス可能な並列形式の信号に変換し、さらにコントローラ630及びユーザターゲット上のマイクロプロセッサからのアクセス要求の調停を行なう。トレースメモリ670はユーザターゲットシステム70上のマイクロプロセッサ10から送られてくるPC情報を格納するためのメモリである。トレースメモリインターフェース660は、ユーザターゲットシステム70上のマイクロプロセッサ10から送られてくるPC情報をトレースメモリ670に格納する。また、コントローラ630からのアクセス要求があった場合に、ユーザターゲット上のマイクロプロセッサから送られてくるPC情報の格納を妨げないように、これを制御する。ランコントロール680は、ユーザターゲットシステム70から供給されるユーザリセット信号RESET\*とユーザシステムの電源ラインの電圧VDDを入力し、デバッグ割込信号DINT\*とデバッグリセット信号DRESET\*をユーザターゲットシステム70

40 【0040】通信インターフェース620はホストコンピュータとの通信を行なう。コントローラ630は、通信インターフェース620を経由してホストコンピュータから送られてきたコマンドを解析し、実行し、結果を送り返す。モニタメモリ640は、モニタプログラムを格納し、これを実行するためのメモリである。モニタメモリインターフェース650は、ユーザターゲットシステム70からの直列形式の信号をモニタメモリ640にアクセス可能な並列形式の信号に変換し、さらにコントローラ630及びユーザターゲット上のマイクロプロセッサからのアクセス要求の調停を行なう。トレースメモリ670はユーザターゲットシステム70上のマイクロプロセッサ10から送られてくるPC情報を格納するためのメモリである。トレースメモリインターフェース660は、ユーザターゲットシステム70上のマイクロプロセッサ10から送られてくるPC情報をトレースメモリ670に格納する。また、コントローラ630からのアクセス要求があった場合に、ユーザターゲット上のマイクロプロセッサから送られてくるPC情報の格納を妨げないように、これを制御する。ランコントロール680は、ユーザターゲットシステム70から供給されるユーザリセット信号RESET\*とユーザシステムの電源ラインの電圧VDDを入力し、デバッグ割込信号DINT\*とデバッグリセット信号DRESET\*をユーザターゲットシステム70

に与えることにより、ユーザプログラムをリセットしたり、停止させたり、あるいは、実行させたりする。ターゲットインターフェース690は、パワーオン時のユーザターゲットシステム70及びデバッグツール60を保護するための回路と、ターゲットユーザシステム70の電源電圧に応じて入出力電圧を調整する回路から成る。

【0041】<デバッグツールマイクロプロセッサ間のインターフェース信号>デバッグツールマイクロプロセッサ間のインターフェース信号は、全部で20本である。内訳を以下に示す。入力／出力は、マイクロプロセッサ側から見た場合の方向を示す。

【0042】デバッグモジュール30とデバッグツール60との外部デバッグインターフェース信号は、次の8本である。

1. DCLK : 出力
2. DRESET : 入力
- 3-5. PCST(2:0) : 出力
6. SDA0 / TPC : 出力
7. SDI / DINT : 入力
8. DBGE : 入力

【0043】デバッグツール60専用ではないが、下記のような信号もデバッグツール60に接続する。

9. RESET : 出力
10. VDD : 出力

【0044】さらに、10本のグランド線を接続する。

- 11-20. GND : グランド

#### 【0045】

(1)DCLK (Debug clock) : 出力端子  
デバッグツール60へのクロック出力である。シリアルモニタバス、PCトレースインターフェースの信号のタイミングは、すべてこのデバッグクロックDCLKで規定される。シリアルモニタバス動作の時は、プロセッサコア20の動作クロックを分周したクロックになる。

【0046】(2)DRESET\* (Debug reset) : 入力端子  
(プルアップ付き端子)

デバッギングリセット入力。ローアクティブの信号である。DRESET\*がアサートされると、ICEモジュールは初期化される(DBGEには無関係)。デバッグツール60を使用しないときには、この端子を未接続にする。

#### 【0047】

(3)PCST(2:0) (PC trace status) : 出力端子  
以下のPCトレースステータス情報とシリアルモニタバスのモードを出力する。以下の表に、この3ビットのPCSTの出力するコードの意味を示す。

#### 【0048】

##### 【表1】

111(STL)	バイブライнстール
110(JMP)	分岐／ジャンプ成立 (PC出力あり)
101(BRT)	分岐／ジャンプ成立 (PC出力なし)
100(EXP)	例外発生 (例外ベクタコード出力あり)

011(SEQ) : シーケンシャル実行 (1命令実行したこと  
を示す)

010(TST) : バイブライнстール時のトレーストリガ  
出力

001(TSQ) : 実行時のトレーストリガ出力

000(DBM) : デバッグモード

(0:ローレベル、1:ハイレベル)

【0049】(4)DBGE\* (Debugger Enable) : 入力端子  
(プルアップ付き端子)

10 デバッグツール60の接続の有無を示す。外部にデバッギングツール60が接続されていない時には、プルアップのため、ハイレベルになる。デバッグツール60側ではローレベルにしてあるので、デバッグツールを接続するとローレベルになる。

【0050】デバッグツール60が接続されていない場合 (DBGE\*信号がハイレベルの場合) には、プロセッサコア20のデバッグ例外ベクタアドレスはユーザに解放されている領域になり、デバッグ例外により、ユーザの作成したモニタなどへ移行することができる。また、ユーザリセットにより、デバッグモジュール機能が初期化され、デバッグ機能はハードウェアブレーク機能を除いてディスエーブルされ、マイクロプロセッサの電力消費を低減させる。また、出力信号 (SDAO/TPC, DCLK, PCST[2:0]) は、すべてハイインピーダンス状態になる。

20 【0051】デバッグツール60が接続されている時には、プロセッサコア20のデバッグ例外ベクタアドレスはユーザに解放されていないモニタ専用領域になる。この時、ユーザリセットでは、デバッグモジュールは初期化されない。これにより、ユーザリセットの直後でもデ

30 バッグ機能が活用できる。つまり、ユーザリセット直後のユーザターゲットシステムの挙動を観測したいという要求に対しても、本発明の構成によって対応することができる。

【0052】(5)SDAO/TPC (Serial Data and address o  
utput/Target PC) : 出力端子

マイクロプロセッサがモニタプログラムを実行している時 (以下、デバッグモードと称する) は、データ／アドレスをシリアルで出力する端子SDAO(Serial Data and Address Output)として、またマイクロプロセッサがユーザプログラムを実行している時 (以下、ノーマルモードと称する) は、ターゲットPCをシリアルで出力する端子TPC(Target PC)として機能する。

【0053】SDAOとしての動作

データ、アドレス、リードライト、バイトイネーブル信号を1ビットづつシリアルに出力する信号端子である。バスサイクル開始前にスタートビットを出力する。(つまり、1クロック間ローレベルを出力する。) 出力の順は、リード時は、スタートビット (ローレベル)、A2-A19, RD, WR, BE0-BE3。ライト時は、スタートビット (ロー  
50 レベル)、A2-A19, RD, WR, BE0-BE3, D0-D31である。

## 【0054】TPCとしての動作

分岐／ジャンプ命令のターゲットアドレスと例外／割込みのベクタ番号を出力するための信号である。ターゲットアドレスは、下位アドレスA[2]から上位アドレスA[31]まで順に出力される。

## 【0055】(6)SDI/DINT\* (Serial Data Input/Debug Interrupt) : 入力端子 (ブルアップ付き)

デバッグモード中はシリアルデータ入力端子SDI (Serial Data input)として、またノーマルモード中はデバッグ割込み端子DINT\* (Debug Interrupt)として動作する。

## 【0056】SDIとしての動作

データ入力信号端子。リードの時には、外部からスタートビット (ローレベル) が入力されると、次のクロックからデータ入力を開始する。ライトのとき、ローレベルが入力されると、バスサイクルが終了する。入力の順は、リード時はスタートビット (ローレベル) 、D[0]-D[31]である。ライト時は終了ビット (ローレベル) だけ入力する。

## 【0057】DINT\*としての動作

デバッグツール60からのデバッグ割込み入力である。デバッグツール60を使用しない時にはこの端子を未接続にする。

## 【0058】(7)RESET\* (Reset) : 出力端子

ユーザリセット端子である。この信号をデバッグツールに接続することにより、例えばデバッグモジュール30からの応答がない時にそれがユーザリセット信号によるものであるかどうかが判断できる。また、ユーザリセット直後にデバッギングリセットをかけたい時には、このRESET\*信号がハイレベルになった後までDRESET\*を入れることで、実現が可能である。

## 【0059】(8)VDD (VDD) : 出力端子

ユーザターゲットシステムの電源ラインである。これをデバッグツール60に入力してやることにより、デバッグツール60は、ユーザターゲットシステム70の電源電圧を知ることができる。これにより、ユーザターゲットシステムの電源電圧に合わせて、入力波形のスレッシュホールド値を変更したり、出力波形の電圧レベルを変更したりすることができる。また、ユーザターゲットシステム70の電源が投入されていないと判断される場合には、デバッグツール60側の出力用デバイスをハイインピーダンスにし、デバイスの保護をはかる。

## 【0060】(9)GND (GND)

デバッグツール60間とユーザターゲットシステム70間のグランドレベルを合わせるために、10本のグランド線を接続する。これらは、デバッグツール60間とユーザターゲットシステム70間の伝送ケーブル中では上記(1)-(8)の信号の間に位置し、信号間のクロストークを低減する役割も果たす。

## 【0061】&lt;シリアルモニタバス回路の詳細説明&gt;図6のシリアルモニタバス回路34の動作について説明す

る。

【0062】[シリアルモニタバスの機能概略] デバッグモード中に、プロセッサコア20がモニタ専用領域をアクセスする場合、シリアルモニタバス回路34を介して、デバッグツール60上のメモリがアクセスされる。

【0063】シリアルモニタバスを用いたライトオペレーションでは、シリアルモニタバス回路34は、アドレス、バス制御信号、データをSDAO信号に、1ビットづつシリアルに出力する。リードオペレーションでは、アド

レス、バス制御信号を1ビットづつシリアルに出力する。リードオペレーションでは、アドレス、バス制御信号をSDAO信号に出力し、SDI信号からデータを1ビットづつシリアルに入力する。

【0064】シリアルモニタバスは、プロセッサコア20の動作クロックCLKを分周したクロックCLK2で動作する。

【0065】シリアルモニタバスに出力するアドレスはプロセッサコアのアドレス信号のA[19:2]の18ビットであり、1Mバイトのメモリ空間をアクセスできる。

【0066】プロセッサコア20のバイトイネーブル信号BE[3:0]をシリアルモニタバスへ出力するので、バイト、ハーフワード、3バイトのアクセスも可能である。ただし、シリアルモニタバスでは、バイト、ハーフワード、3バイトアクセスの場合も、32ビット分のデータを転送する。バイト、ハーフワード、3バイトのライトのとき、BE[3:0]\*がインアクティブなバイト位置のデータは不定である。リードの時は、インアクティブバイトなバイト位置のデータはプロセッサコア20で無視され、リードされない。

【0067】ノーマルモード中は、モニタ専用領域へのライトは無視され、リードは結果が不定となる。このようなライトアクセスがあった時、シリアルモニタバス回路34は、プロセッサコア20へ、バスオペレーションの完了を示すアノリッジ信号を送り、バスオペレーションを終了する。

【0068】<シリアルモニタバスの信号伝送方法の詳細説明>図8にシリアルモニタバスに関わる回路の概略図を示す。これを用いて、信号伝送の手順を説明する。

【0069】[プロセッサコア20がメモリリードを行う場合]

(1) プロセッサコア20から出力される並列形式のアドレス、リード信号、バイトイネーブルをシリアル出力回路A342が直列形式に変換して、SDAOから出力する。

(2) デバッグツール60内のシリアルモニタ入力回路B651はこれを入力し、並列形式に変換し、モニタメモリ640に出力する。

(3) メモリから出力された並列形式のデータは、シリアル出力回路B652で直列形式に変換され、SDIを通じて出力される。

(4) デバッグモジュール30内のシリアル入力回路A343はこれを並列形式に変換して、プロセッサコア20へ出力する。

(5) プロセッサコア20がその並列形式のデータをリードする。

【0070】 [プロセッサコア20がメモリライトを行う場合]

(1) プロセッサコア20から出力される並列形式のアドレス、ライト信号、バイトイネーブル、データをシリアル出力回路A342が直列形式に変換して、SDAOから出力する。

(2) デバッグツール60内のシリアル入力回路B651はこれを入力し、並列形式に変換し、モニタメモリ640に出力する。

(3) モニタメモリ640への書き込みが終了したら、シリアル出力回路B652で、SDIに1クロック間だけローレベルを出力する。

(4) ユーザーターゲットシステム70内のマイクロプロセッサ10中のシリアル入力回路A343は、このローレベルを入力したら、プロセッサコア20へライトサイクル終了したことを伝える。

(5) プロセッサコア20は、ライトサイクルを終了する。

【0071】 <シリアルモニタバスオペレーションのタイミングの詳細説明>以下にシリアルモニタバスオペレーションをタイミングチャートを用いて詳しく説明する。

【0072】 [シリアルモニタバスのリードバスオペレーション] 図9にシリアルモニタバスのリードバスオペレーションのタイミングチャートを示す。

(1) プロセッサコア20がモニタ専用領域に対するリードバスオペレーションを開始する(サイクル1)。プロセッサコア20は、プロセッサバスにアクセスするアドレスを並列形式で出力し、リード信号をアサートし、またリードするバイト位置のバイトイネーブル信号をアサートする。

(2) シリアルモニタバス回路34は、モニタ領域へのリードバスオペレーションの開始を認識すると、SDAO信号にコアクロックCLKを分周したCLK2信号の1クロック間ローレベルを出力する(サイクル2)。

(3) シリアルモニタバス回路34は、プロセッサコア20のリードバスオペレーションで出力された、アドレスA[2]-A[19]、ハイレベル(リードを示す)及びバイトイネーブル信号BE[0]\*-BE[3]\*を、この順でSDAO信号にCLK2信号の1クロック間づつ出力する(サイクル3~25)。

(4) デバッグツール60内のモニタメモリインターフェース650は、SDAO信号に出力されたアドレスA[2]-A[19]、ハイレベル(リードを示す)及びバイトイネーブル信号BE[0]\*-BE[3]\*を、この順で、DCLKに従って1ク

ロックづつ入力し、アドレス及びバイトイネーブル信号を並列形式に変換し、モニタメモリ640に対して出力する。

(5) モニタメモリインターフェース650は、モニタメモリ640から出力された並列形式のデータを直列形式に変換する。データを外部へ出力する前に、1クロック間SDI信号にローレベルを出力する(サイクルn)。続けて、DCLKに同期して、データをD[0]から順に1ビットづつD[31]まで出力する(サイクルn+1~n+32)。

(6) シリアルモニタバス回路34は、SDIにローレベルを検出すると(サイクルn)、次のサイクルよりデータD[0]-D[31]をDCLKの1クロック毎に読み込む(サイクルn+1~n+32)。

(7) シリアルモニタバス回路34は、プロセッサコア20のリードバスの応答信号をアサートし、読み込んだ32ビットのデータをプロセッサバスへ並列形式にして出力する(サイクルn+33)。

(8) プロセッサコア20は、プロセッサバス上のデータを読み込み、リードバスオペレーションを終了する。

【0073】 [シリアルモニタバスのライトバスオペレーション] 図10にシリアルモニタバスのライトバスオペレーションのタイミングチャートを示す。

(1) プロセッサコア20がモニタ専用領域に対するライトバスオペレーションを開始する(サイクル1)。プロセッサコア20は、プロセッサバスにアクセスするアドレスを出力し、ライト信号をアサートする。ライトするバイトの位置のバイトイネーブル信号をアサートする。

(2) シリアルモニタバス回路34は、モニタ領域へのライトバスオペレーションの開始を認識すると、SDAO信号にコアクロックCLKを分周したCLK2信号の1クロック間ローレベルを出力する(サイクル2)。

(3) シリアルモニタバス回路34はプロセッサコア20のライトバスオペレーションで出力された、アドレスA[2]-A[19]、ローレベル(ライトを示す)、バイトイネーブル信号BE[0]\*-BE[3]\*及びライトデータDOUT[0]-DOUT[31]を、この順でSDAO信号にCLK2信号の1クロック間づつ出力する(サイクル3~57)。

(4) デバッグツール60内のモニタメモリインターフェース650は、SDAO信号に出力されたアドレスA[2]-A[19]、ハイレベル(リードを示す)、バイトイネーブル信号BE[0]\*-BE[3]\*及びライトデータDOUT[0]-DOUT[31]をこの順でDCLKに従って、1クロックづつ入力し、アドレス、バイトイネーブル信号、ライトデータを並列形式に変換し、モニタメモリ640に対して出力する。

(5) モニタメモリインターフェース650は、モニタメモリ640への書き込みが終したら、1クロック間SDI信号にローレベルを出力する(サイクルn)。

(6) シリアルモニタバス回路34は、SDIにローレ

ルを検出すると、プロセッサコア20に対して、ライトバス応答信号をアサートする（サイクルn+1）。

(7) プロセッサコア20は、ライトバスオペレーションを終了する。

【0074】<PCトレース回路>ここで、インダイレクトジャンプ、ダイレクトジャンプ、分岐を以下のように定義する。

・インダイレクトジャンプ

レジスタに格納されているアドレスへのジャンプなど、ジャンプ先がその命令自体では特定できないジャンプ。

・ダイレクトジャンプ

命令自身が格納されているアドレス及び命令コードによってジャンプ先のアドレスが特定できるジャンプ。

・分岐

命令自身が格納されているアドレス及び命令コードの一部の和で分岐先が特定できるジャンプ。分岐において、実際にジャンプが発生するかどうかは条件により決まる。実際にジャンプが行なわれることを分岐成立、行なわれなかつたことを分岐不成立と呼ぶ。

【0075】また、PCトレースには、次の2種類のトレースモードがある。

・リアルタイムトレースモード

このモードでは、プロセッサコア20の実行は常にリアルタイムで行なわれるが、インダイレクトジャンプのターゲットPC出力中に次のインダイレクトジャンプが発生した場合、先に発生したインダイレクトジャンプのターゲットPCの出力は中止され、新しいターゲットPCの出力が開始される。

・非リアルタイムトレースモード

このモードでは、上記のようにインダイレクトジャンプが接近して発生した場合、先に発生したインダイレクトジャンプのターゲットPCを出力し終るまでプロセッサコア20のパイプライン処理を停止させる。これにより、プロセッサコア20の実行のリアルタイム性は損なわれるが、インダイレクトジャンプのターゲットPCは必ず完全に出力される。

【0076】PCトレース回路32は、プロセッサコア20から以下の信号を入力する。

・デバッグモード信号

プロセッサコア20がデバッグモード中か、ノーマルモード中かを示す。

・パイプライン実行信号

命令が一命令実行されたことを示す。

・30ビットのターゲットPC[31:0]信号

分岐命令、ジャンプ命令のターゲットアドレス、あるいは例外のベクタアドレスを示す。以下のインダイレクトジャンプ信号、ダイレクトジャンプ命令、分岐成立信号、例外発生信号がアサートされているときに有効である。

・インダイレクトジャンプ信号

インダイレクトジャンプが実行されたことを示す。

・ダイレクトジャンプ信号

ダイレクトジャンプが実行されたことを示す。

・分岐成立信号

分岐成立の分岐命令が実行されたことを示す。

・例外発生信号

例外が発生したことを示す。

【0077】また、PCトレース回路32は、PCトレースを完全に行なうために、以下の信号をプロセッサコア20へ出力する。

・パイプラインストール要求信号

ターゲットPCの出力を完全に行なう非リアルタイムトレースモードの時、プロセッサコア20のパイプラインをストールさせるための信号である。PCトレース回路32は、インダイレクトジャンプのターゲットPCを出力している時に次のインダイレクトジャンプが発生すると、この信号をアサートしてプロセッサコア20のパイプラインをストールさせる。出力中のターゲットPCが完全に出力されるとこの信号をネゲートして、プロセッサコア20のパイプライン処理を再開させる。

【0078】PCトレース回路32は、命令/データアドレスブレーク回路31とプロセッサブレーク回路33からトリガ要求信号を入力する。また、レジスタ回路35内のレジスタに割り当てられているトレースモードを切替えるビットの状態を入力する。

【0079】PCトレース回路32は、ノーマルモード時にプロセッサコア20が出力するPCトレース情報を、1ビットのPC出力(TPC信号)と、3ビットのステータス信号(PCST[2:0]信号)にしてデバッグツール60へ出力する。以下にPCST[2:0]とTPC信号について説明する。

【0080】[PCST[2:0]] 各クロック毎に、命令の実行状態をPCST[2:0]に出力する。下記の説明では、“0”はローレベルを、“1”はハイレベルを表す。

・111(STL):パイプラインストール

トレーストリガ出力のない状態で、命令の実行終了がなかったことを示す。

・110(JMP):分岐/ジャンプ成立 (PC出力あり)

分岐成立の分岐命令、ジャンプ命令の実行が終了し、TPC信号にそのターゲットアドレス(分岐、ジャンプ先のアドレス)の出力を開始したことを示す。

・101(BRT):分岐/ジャンプ成立 (PC出力なし)

分岐成立の分岐命令、ジャンプ命令の実行が終了したが、TPC信号にそのターゲットアドレス(分岐、ジャンプ先のアドレス)の出力がないことを示す。

・100(EXP):例外発生 (例外ベクタのコード出力あり)

例外が発生したことを示す。同時にTPC信号へ例外ベクタのコード出力が開始されたことを示す。コードは、3ビットあり、下位code(0)からcode(1)、code(2)の順でTPC信号へ出力する。

## 【0081】

例外の種類	ベクタアドレス	code
リセット, Nmi	BFC0_0000	(100) 4
UTLB(BEV=0)	8000_0000	(000) 0
UTLB(BEV=1)	BFC0_0100	(110) 6
その他(BEV=0)	8000_0080	(001) 1
その他(BEV=1)	BFC0_0180	(111) 7

ここで、BEVはレジスタ回路35内のあるレジスタ内の1ビットであり、その値によって、例外処理のベクタアドレスを変更できる。

【0082】・011(SEQ):シーケンシャル実行(1命令実行したことを示す)

ジャンプ、分岐の実行(JMP, BRT)以外、あるいはトレーストリガ出力要求(TSQ)のない状態で、命令が実行されたことを示す。分岐不成立の分岐命令が実行されたときもこのコードが出力される。

・010(TST):パイプラインストール時のトレーストリガ出力

命令の実行終了がないクロックで、命令アドレストレーストリガまたは、プロセッサバストレーストリガが発生したことを示す。

・001(TSQ):実行時のトレーストリガ出力

命令の実行が終了したクロックで、命令アドレストレーストリガあるいはプロセッサバストレーストリガが発生したことを示す。

・000(DBM):デバッグモード

ノーマルモードでは、このコードが出力されることはない。

【TPC】分岐命令、ジャンプ命令のターゲットアドレスを出力する信号である。PCST[2:0]に110(JMP)が出力されたクロックから、ターゲットアドレスの出力を開始する。ターゲットアドレスは、下位A(2)から1クロック毎に出力される。PCST[2:0]に、100(EXP)が出力されたクロックから、例外ベクタの3ビットコードを出力する。コードは、下位code(0)から1クロック毎に出力する。

【0083】TPC信号によって、1ビットシリアルでターゲットアドレスを出力するため、前の分岐ジャンプ命令をTPC信号へ出力している途中に、次の分岐、ジャンプ命令あるいは例外が発生することがある。この場合のTPCへのターゲットアドレス出力の優先度について、以下のように規定する。

【0084】(1)トレースモードがリアルタイムモードである場合、ターゲットPC出力中に新たにインダイレクトジャンプが発生すると、その時のターゲットPC出力を終了し、必ず新たなインダイレクトジャンプのターゲットPCの出力を開始する。

(2)トレースモードが非リアルタイムモードである場合、ターゲットPC出力中に新たにインダイレクトジャンプが発生すると、その時のターゲットPC出力を終了するまでプロセッサコアのパイプライン処理を停止し、

必ずそのターゲットPC出力を終了してからプロセッサコアのパイプライン処理を再開し、新たなインダイレクトジャンプのターゲットPCの出力を開始する。

(3)ターゲットPC出力中に例外が発生したときは、例外のベクタ番号(3ビット)を必ず出力し、その後、中断していたPC出力を再開する。

(4)ターゲットPC出力中に、新たにダイレクトジャンプ、分岐が発生したときは、そのダイレクトジャンプ、分岐のターゲットアドレスの出力は行なわない。ダイレクトジャンプ、分岐については、発生時にターゲットPCを出力していないのみ、そのターゲットPCが

10 出力される。ダイレクトジャンプまたは分岐の場合には、ターゲットアドレスが出力されなくても、その命令がアドレス何番地にストアされているのかがわかれれば、メモリにストアされているその命令のコードを参照することにより、ジャンプ先、分岐先のアドレスが判明する。当該命令がアドレス何番地にストアされているかは、その前に発生したダイレクトジャンプまたは分岐から何クロック目に実行されているかで判明する。

【0085】次にPCトレースの出力例を図を用いて説明する。

【0086】(例1)分岐命令のPCトレース

図11に分岐命令のPCトレース出力の例を示す。最初の分岐成立の分岐命令beqでは、TPCにターゲットPCを出力していないので、PCST[2:0]にJMPコードを出力し、TPCにターゲットPCの出力を開始する。分岐不成立の分岐命令bneでは、PCST[2:0]にSEQコードを出力する。2番目の分岐成立の分岐命令bneはダイレクトジャンプであり、最初の分岐命令のターゲットPCを出力しているので、TPCにターゲットPCを出力しない。PCST[2:0]には、BRTコードを出力する。

【0087】(例2)インダイレクトジャンプ命令のPCトレース

図12にインダイレクトジャンプ命令のPCトレース出力の例を示す。最初のインダイレクトジャンプ命令jr1では、PCST[2:0]にJMPコードを出力し、TPCにターゲットPCの出力を開始する。分岐不成立の分岐命令bneでは、PCST[2:0]にSEQコードを出力する。2番目のインダイレクトジャンプ命令jr2では、最初のインダイレクトジャンプ命令のターゲットPCの出力を中止して、jr2のターゲットPCをTPCへ出力する。PCST[2:0]にはJMPコードを出力する。

【0088】(例3)例外とインダイレクトジャンプ命令のPCトレース

図13に例外とインダイレクトジャンプ命令のPCトレース出力の例を示す。ソフトウェアブレーク命令break例外が発生すると、PCST[2:0]にEXPコードを出力し、TPCに例外ベクタコードの出力を開始する。分岐不成立の分岐命令bneでは、PCST[2:0]にSEQコードを出力する。50 インダイレクトジャンプ命令jr2では、jr2のターゲット

PCをTPCへ出力し、PCST[2:0]には、JMPコードを出力する。

**【0089】(例4) デバッグ例外発生時にPCを出力していない場合のPCトレース**

図14にデバッグ例外発生時のPCST[2:0]の出力タイミングの例を示す。同図において、DM信号は、ハイレベルでデバッグモード中であることを示し、ローレベルでノーマルモード中であることを示す、プロセッサコア20中の内部信号である。プロセッサコア20がデバッグ例外あるいはデバッグリセットを発生すると、デバッグモードに入る。このとき、PCトレース回路32はPCST[2:0]出力にDBMコードを出力する。ターゲットPCを出力していないときは、デバッグ例外を発生した命令の実行終了直後にデバッグモードになる。デバッグ例外を発生した直前の命令までのPCトレース情報が出力される。

**【0090】(例5) デバッグ例外発生時にPCを出力している場合のPCトレース**

図15にデバッグ例外発生時に、ターゲットPCを出力している場合のPCST[2:0]出力タイミングを示す。ターゲットPCを出力している時は、そのターゲットPCが完了した後、デバッグモードになる。デバッグ例外を発生した直前の命令までのPCトレース情報が出力される。ターゲットPCを出力している間は、PCST[2:0]にはSTLが出力される。

**【0091】(例6) デバッグモードからノーマルモードへの移行時のPCトレース**

図16にデバッグモードからの復帰時のPCST[2:0]の出力タイミングを示す。デバッグ例外、デバッグリセットからの復帰命令DERET命令の分岐ディレイスロット命令までがデバッグモードに属する。DERET命令の戻り先の命令からノーマルモードになり、PCトレースが有効になる。

**【0092】<トレーストリガの詳細説明>PCST[2:0]信号へのトレーストリガの出力について説明する。命令アドレストレーストリガ、データアドレストレーストリガ、プロセッサバストレーストリガのいずれかが発生した場合、次のような論理でトレーストリガ情報がPCST[2:0]に出力される。**

(1) その時に分岐成立の分岐命令もしくはジャンプ命令が実行されているか、例外が発生している場合

ここにおいてもしトレーストリガが発生しなければ、PCST[2:0]にはJMP、BRT、EXPコードが出力されるはずである。この(1)の場合、にはトレーストリガが発生してもPCST[2:0]の出力は変化せずに保留され、直後の

(2) または(3)のケースで、トレーストリガ情報が出力される。

(2) その時にパイプラインがストールしている場合

ここにおいてもしトレーストリガが発生しなければ、PCST[2:0]には、STLコードが出力されるはずである。この

(2) の場合、トレーストリガが発生すると、PCST[2:0]にはTSTコードが出力される。

(3) (1) 及び (2) 以外の場合、すなわちパイプラインが順次実行をしている場合

ここにおいてもしトレーストリガが発生しなければ、PCST[2:0]にはSEQコードが出力されるはずである。この

(3) の場合、トレーストリガが発生すると、PCST[2:0]にはTSQコードが出力される。

**【0093】以下に実際の例を波形図にて示す。**

**【0094】(例1) トレーストリガの発生例**

図17に、通常の命令を順次実行中のトレーストリガの発生例を示す。add命令実行中にトレーストリガが発生したので、トレーストリガのコードTSQを出力する。

**【0095】(例2) 例外発生命令実行中にトレーストリガが発生した場合の例**

図18に、例外発生命令実行中にトレーストリガが発生した例を示す。ソフトウェアブレーク命令break実行中にトレーストリガが発生したが、break命令実行のクロックではPCST[2:0]信号に例外発生のコードEXPを出力し、次のクロックでトレーストリガのコードを出力する。この例ではストール状態なので、TSTコードを出力する。

**【0096】(例3) インダイレクトジャンプ命令実行中にトレーストリガが発生した場合の例**

図19に、インダイレクトジャンプ命令実行中にトレーストリガが発生した場合の例を示す。インダイレクトジャンプ命令jr2を実行中にトレーストリガが発生したが、jr2命令実行のクロックではPCST[2:0]信号にジャンプのコードJMPを出力し、次のクロックでトレーストリガのコードを出力する。この例では命令実行状態なので、TSQコードを出力する。

**【0097】<トレースメモリインターフェース回路の説明>**図20に、デバッグツール60内のトレースメモリインターフェース660とトレースメモリ670の構成図を示す。

**【0098】**デバッグモジュール60から出力されたTPC及びPCST[2:0]は、トレースデータレジスタ663を通してトレースメモリ670へ書き込まれる。この時のアドレスの値は、トレースアドレスカウンタ662から供給される。PCST[2:0]はトレーストリガデコーダ664にも入力されており、トレーストリガの発生をトレースメモリコントロール回路661へ伝える。トレースアドレスカウンタの初期値の設定、インクリメント／停止の指示は、トレーストリガデコーダ664の出力結果に基づいてトレースメモリコントロール回路661が行なう。

**【0099】**コントローラ630がトレースメモリ670の内容を読み出す場合には、コントローラアドレスレジスにアドレスを設定してトレースメモリコントロール回路661にリード要求を出すと、データがコントロー

データレジスタ667に書き込まれるので、これを読み出す。

【0100】コントローラ630がトレースメモリ670へデータを書き込む場合には、コントローラアドレスレジスタ665にアドレスを設定し、コントローラデータレジスタ667にデータを設定し、トレースメモリコントロール回路661にライト要求を出す。

【0101】<低消費電力化の詳細説明>図21に、デバッグモジュール内の電源供給の構造を示す。

【0102】デバッグツールが接続されていない場合、信号DBGE\*はハイレベルになっている。ユーザリセット時にこのDBGE\*信号がハイレベルの場合には、電源供給スイッチ38が切断され、シリアルモニタバス回路34、PCトレース回路32には、電源が供給されなくなる。これらの回路は、外部デバッグツールが接続されていない時には使用する意味がないので、電源を供給しないことにより、マイクロプロセッサ全体での消費電力を低減することができる。

【0103】デバッグツールが接続されていない場合でも、命令/データアドレスブレーク回路31、プロセッサバスブレーク回路33及びレジスタ回路35には電源が供給され、命令/データアドレスブレークとプロセッサバスブレークの機能は機能する。デバッグ例外のベクタアドレスをユーザ領域に変更することにより、これらのハードウェアブレーク機能をユーザがデバッグの用途に使用できる。

【0104】<デバッグモジュール初期化回路の詳細説明>図22に、デバッグモジュール初期化回路を示す。

【0105】デバッグモジュールが接続されていない時にはDBGE\*がハイレベルになるので、ユーザリセットがアサートされると、デバッグモジュール初期化信号がアサートされ、デバッグモジュール30が初期化される。

【0106】デバッグツールが接続されていない場合でも、電源ラインへのノイズなどが原因でデバッグモジュール自体が誤動作してしまい、プロセッサコア20へ割込みを要求したりする可能性がある。デバッグツールは接続されていないので、デバッグリセット(DRESET\*)信号を入れることはできない。ユーザリセットではデバッグモジュールを初期化できないと、このデバッグモジュールを初期化する手段がない。従って、デバッグツールが接続されていない場合に、ユーザリセットがデバッグモジュールを初期化することは極めて重要である。

【0107】逆に、デバッグツールが接続されている場合には、DBGE\*がローレベルになるので、ユーザリセットがアサートされても、デバッグモジュール初期化信号はアサートされず、デバッグモジュール30が初期化されない。

【0108】この動作は、ユーザリセットの直後にトレーストリガをかけたい場合などに極めて有効である。もしユーザリセットでデバッグモジュール30が初期化さ

れてしまうと、例えばトレーストリガをかけるためにはいったんデバッグモードに入りて必要なレジスタを設定し直さなければならない。しかし、デバッグモードに入ることで、その間、実時間動作ができなくなってしまうので、とらえようとしている現象がとらえられない可能性が高い。

【0109】電源ラインへのノイズなどが原因でデバッグモジュール自体が誤動作してしまった場合でも、デバッグツールが接続してある場合、デバッグツールからの10デバッグリセット(DRESET\*)で初期化できるので問題ない。

#### 【0110】

【効果】以上詳細に説明したように、本発明によれば、従来例に比べて、次のような効果が得られる。

【0111】従来例1に比べて、次のような効果が得られる。

- ・モニタプログラムをデバッグツール側のメモリで走らせるため、ユーザのメモリを使用しない。

- ・モニタへ入るために専用のデバッグ例外を用意しているため、ユーザの割込みに制限を加えない。

- ・ユーザターゲットシステムにシリアルインターフェースを用意する必要がない。

- ・ハードウェアブレークポイントが使用可能である。

【0112】従来例2に比べて、次のような効果が得られる。

- ・マイクロプロセッサ内部にシーケンサをもつ必要がないため、マイクロプロセッサ内部に追加されるデバッグのためのロジック回路が簡単である。

- ・モニタプログラムによって、レジスタにアクセスするため、派生品のマイクロプロセッサでレジスタが追加されるような場合でも、モニタを変更するだけで容易にそれらに対するアクセスが可能になる。

- ・上記2項の理由により、マイクロプロセッサコアに多種類の周辺回路を付け加えている場合でも、デバッグのためのロジック回路を共通化できる。この共通モジュールを周辺回路の一部としてマイクロプロセッサに内蔵することで、プロセッサコアが共通で周辺回路が異なる多様な派生マイクロプロセッサに対して共通のデバッグツールを適用することが可能となる。

【0113】従来例3に比べて、次のような効果が得られる。

- ・デバッグツールのハードウェア仕様を共通化できる。
- ・デバッグツールとの接続のための信号の本数を減らせる。

- ・上記理由により、プローブを小型化、低価格化できる。

- ・ユーザターゲット上のマイクロプロセッサがメモリやI/Oをアクセスするので、デバッグツールに要求されるタイミング条件が改善される。

- ・デバッグツールに接続されない信号に関しては、影響

を及ぼさない。

- ・デバッグツールとマイクロプロセッサ間の通信速度を必要に応じて、遅くできる。
- ・上記理由により、高速のマイクロプロセッサにも対応できる。

【0114】従来例4に比べて、次のような効果が得られる。

- ・デバッグツールを用いて、ターゲットメモリやI/Oへのアクセス及び実行制御が実現される。
- ・キャッシングメモリで実行された命令についても、そのアドレス情報がトレースできる。 10

【図面の簡単な説明】

【図1】従来例1の構成を示す図。

【図2】従来例2の構成を示す図。

【図3】従来例3の構成を示す図。

【図4】従来例4の構成を示す図。

【図5】本発明の一実施例の構成を示す図。

【図6】本発明の一実施例中のデバッグモジュールの構成を示す図。 20

【図7】本発明の一実施例中のデバッグツールの構成を示す図。

【図8】本発明の一実施例中のシリアルモニタバスに関する回路の概略図を示す図。

【図9】本願の一実施例中のシリアルモニタバスのライトバスオペレーションのタイミングチャート。

【図10】本願の一実施例中のシリアルモニタバスのライトバスオペレーションのタイミングチャート。

【図11】本願の一実施例における分岐命令のPCトレース出力の例を示すタイミングチャート。 30

【図12】本願の一実施例におけるインダイレクトジャンプ命令のPCトレース出力の例を示すタイミングチャート。

【図13】本願の一実施例における例外とインダイレクトジャンプ命令のPCトレース出力の例を示すタイミングチャート。

【図14】本願の一実施例におけるデバッグ例外発生時のPCST[2:0]の出力タイミングの例を示すタイミングチャート。

【図15】本願の一実施例において、デバッグ例外発生時にターゲットPCを出力している場合のPCST[2:0]出力タイミングを示すタイミングチャート。 40

【図16】本願の一実施例におけるデバッグモードからの復帰時のPCST[2:0]の出力タイミングを示すタイミングチャート。

【図17】通常の命令を順次実行中のトレーストリガの発生の例のタイミングチャート。

【図18】例外発生命令実行中にトレーストリガが発生

した場合の例のタイミングチャート。

【図19】インダイレクトジャンプ命令実行中にトレーストリガが発生した場合の例のタイミングチャート。

【図20】デバッグツール60内のトレースメモリインターフェースとトレースメモリの構成を示す図。

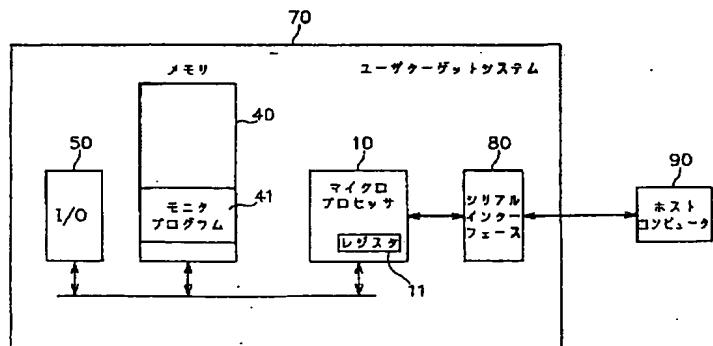
【図21】デバッグモジュール内の電源供給部の構成を示す図。

【図22】デバッグモジュール初期化回路の構成を示す図。

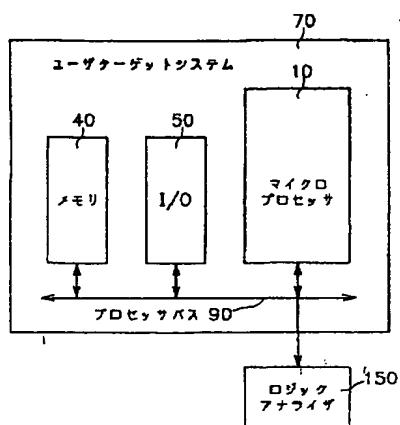
【符号の説明】

10 : マイクロプロセッサ  
 15 : 内部デバッグインターフェース  
 16 : 内部プロセッサバス  
 20 : プロセッサコア  
 30 : デバッグモジュール  
 31 : 命令/データアドレスブレーク回路  
 32 : PCトレース回路  
 33 : プロセッサバスブレーク回路  
 34 : シリアルモニタバス回路  
 35 : レジスタ回路  
 36 : 外部インターフェース回路  
 37 : 分周回路  
 38 : 電源供給スイッチ  
 40 : メモリ  
 50 : I/O  
 60 : デバッグツール  
 70 : ユーザターゲットシステム  
 71 : 外部デバッグインターフェース  
 90 : プロセッサバス  
 342 : シリアル出力回路A  
 343 : シリアル入力回路A  
 620 : 通信インターフェース  
 630 : コントローラ  
 640 : モニタメモリ  
 650 : モニタメモリインターフェース  
 651 : シリアル入力回路B  
 652 : シリアル出力回路B  
 660 : トレースメモリインターフェース  
 661 : トレースメモリコントロール回路  
 662 : トレースアドレスカウンタ  
 663 : トレースデータレジスタ  
 664 : トレーストリガデコーダ  
 665 : コントローラアドレスレジスタ  
 667 : コントローラデータレジスタ  
 670 : トレースメモリ  
 680 : ランコントロール  
 690 : ターゲットインターフェース

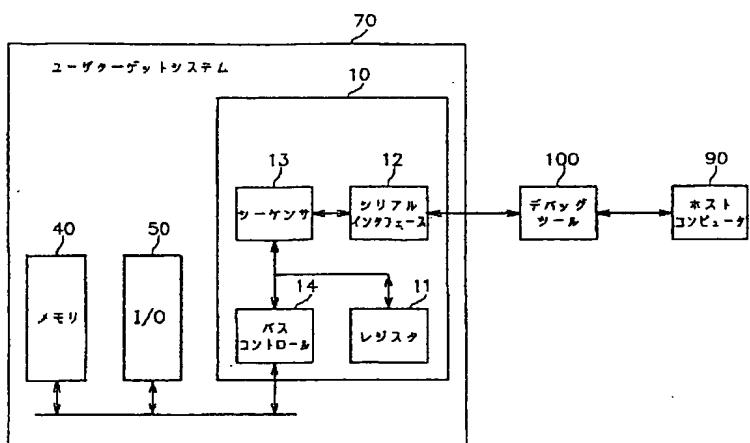
【図1】



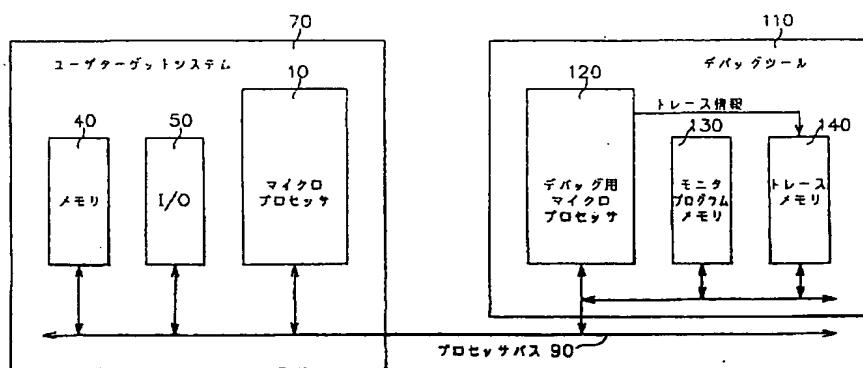
【図4】



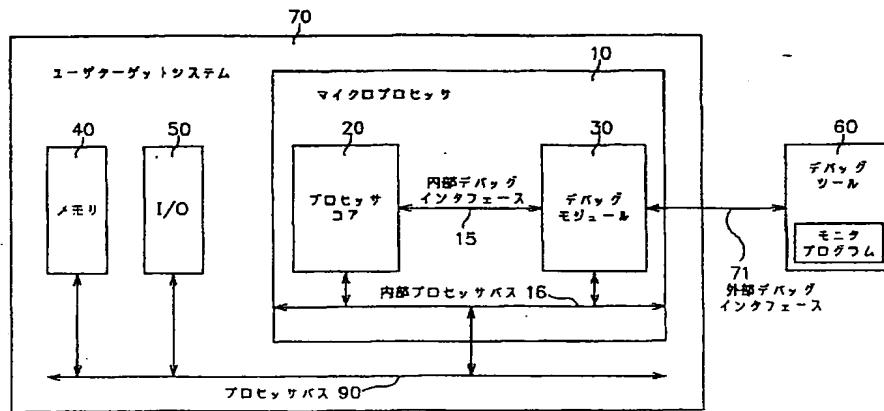
【図2】



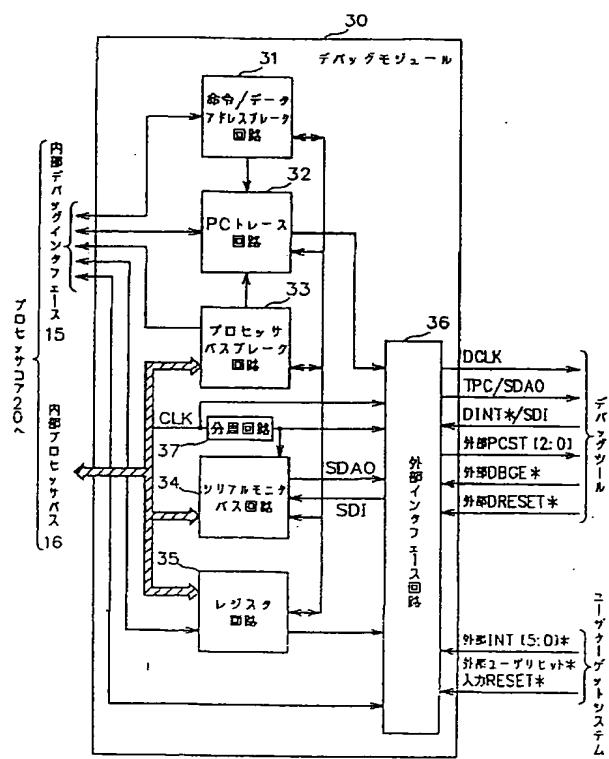
【図3】



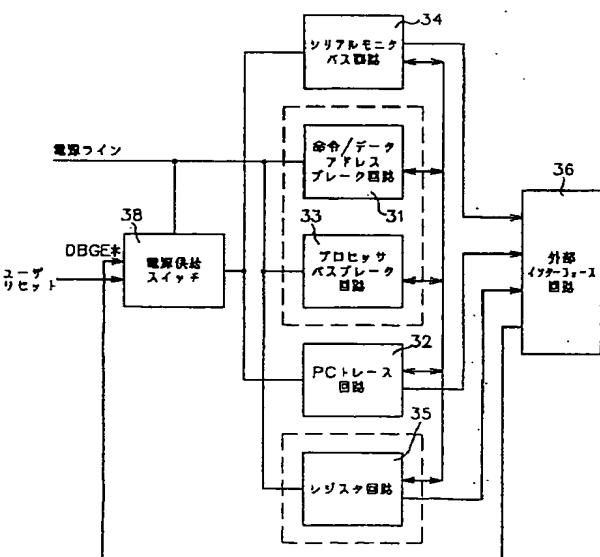
【図5】



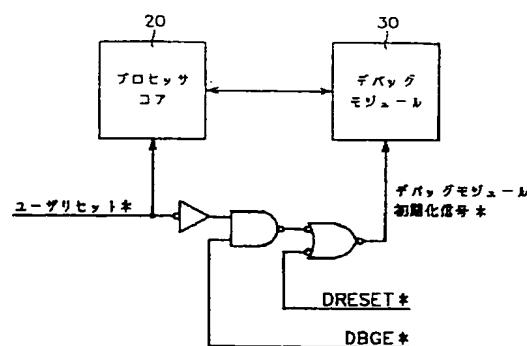
【図6】



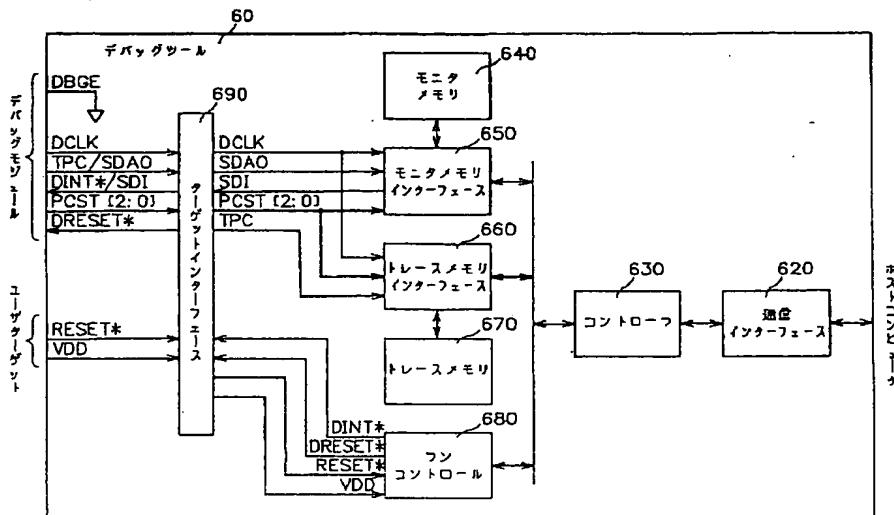
【図21】



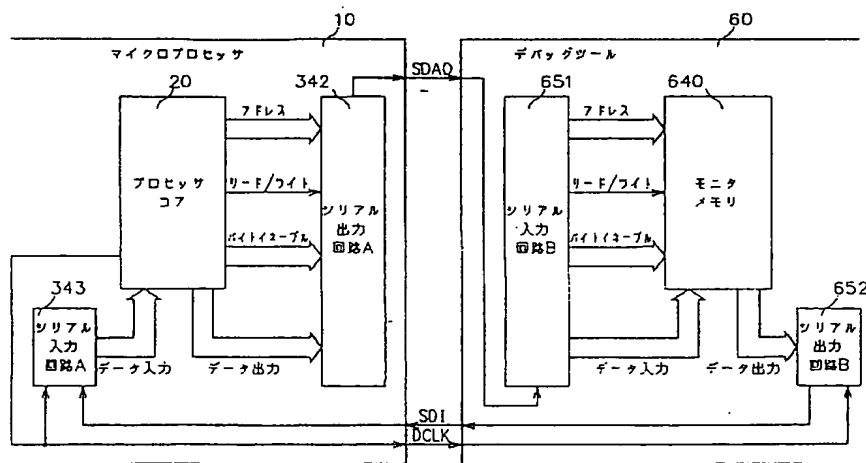
【図22】



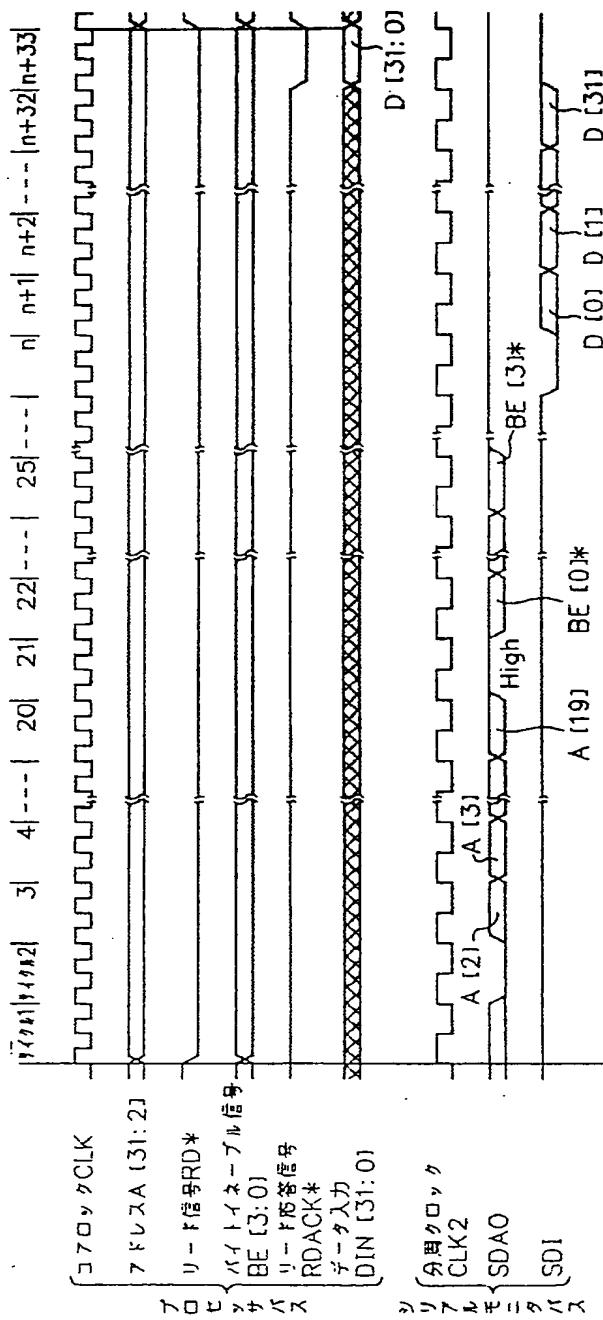
【図7】



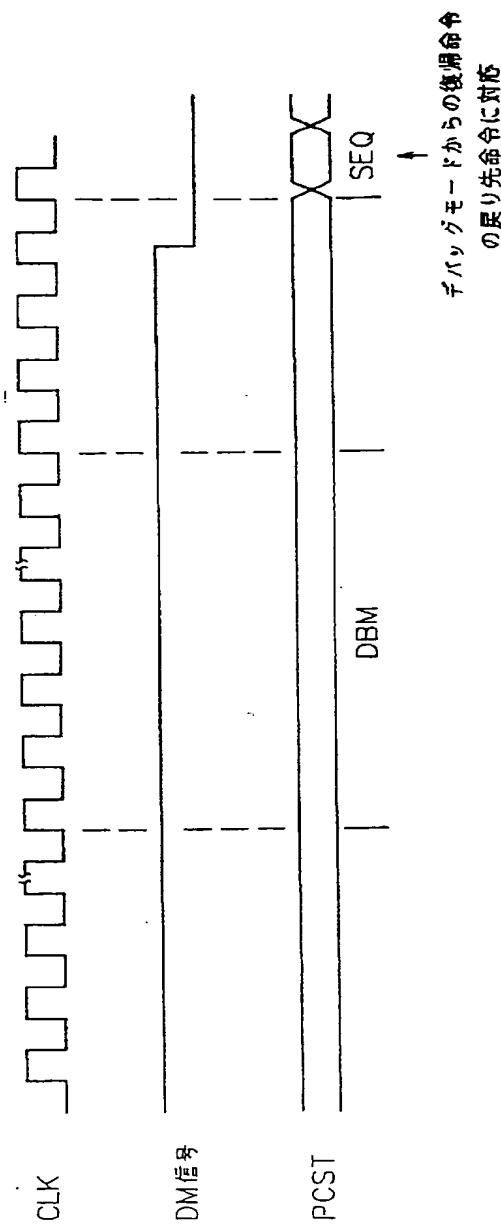
【図8】



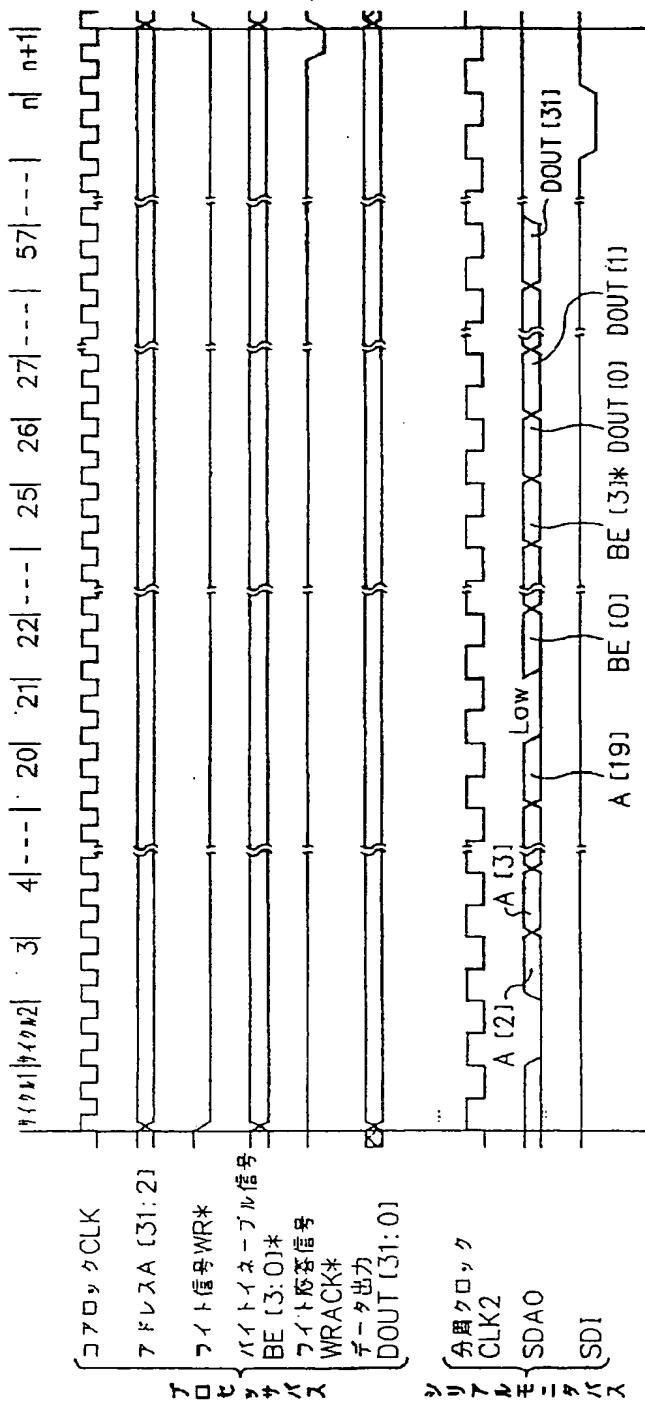
【図9】



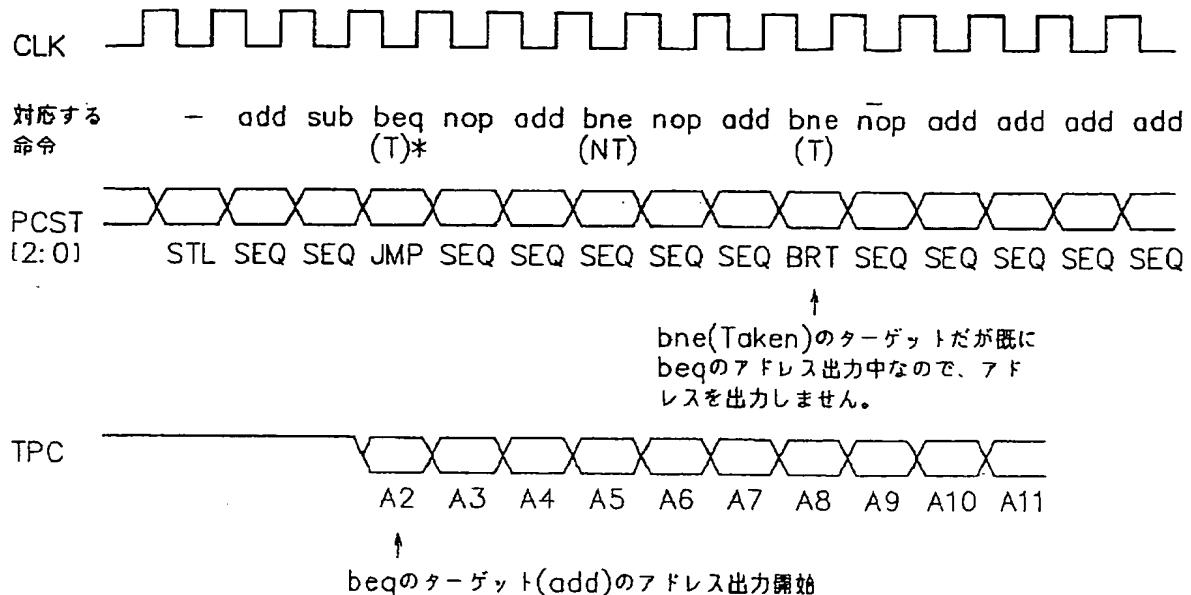
【図16】



【図10】

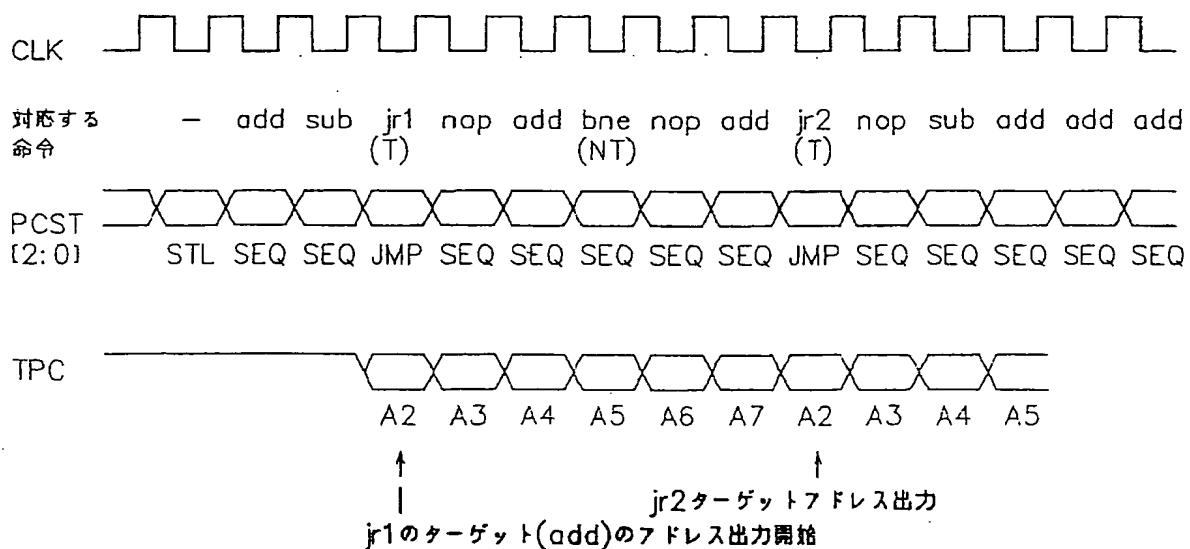


【図11】

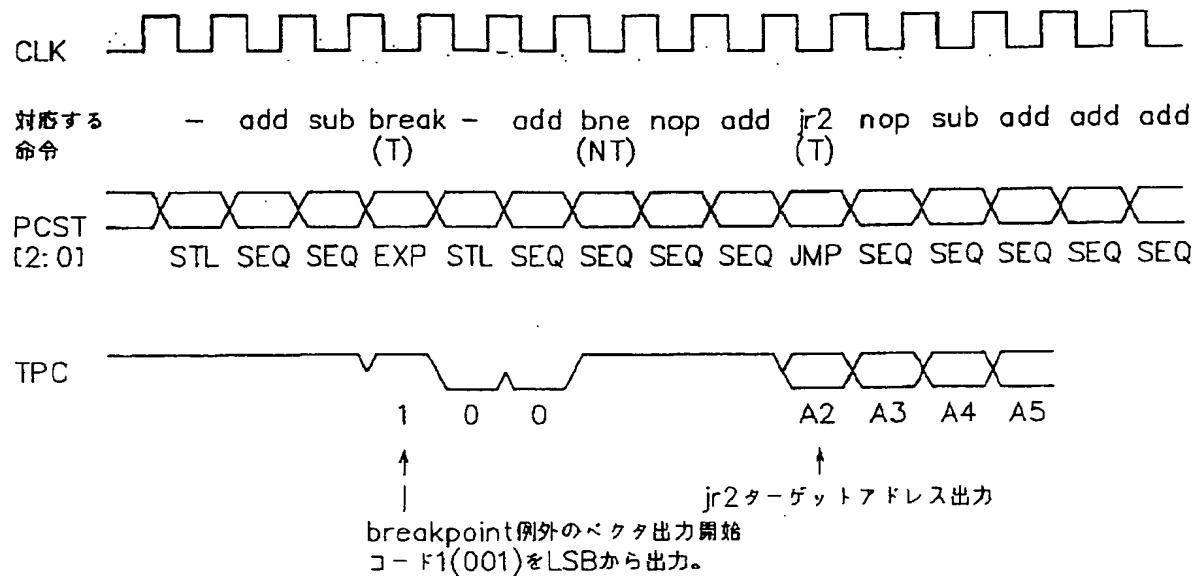


\* : T: Taken(分岐成立)、 NT: Not Taken(分岐不成立)を示します。

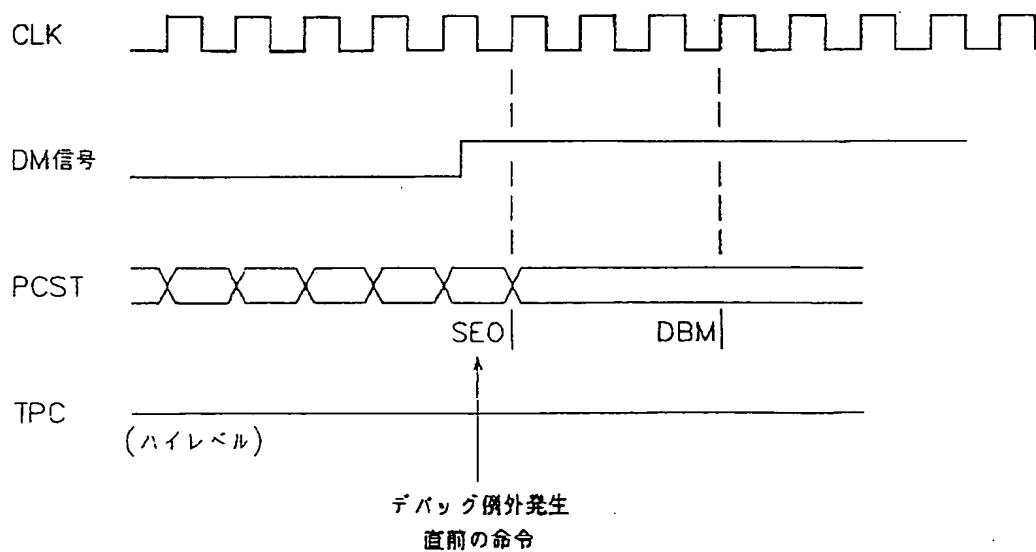
【図12】



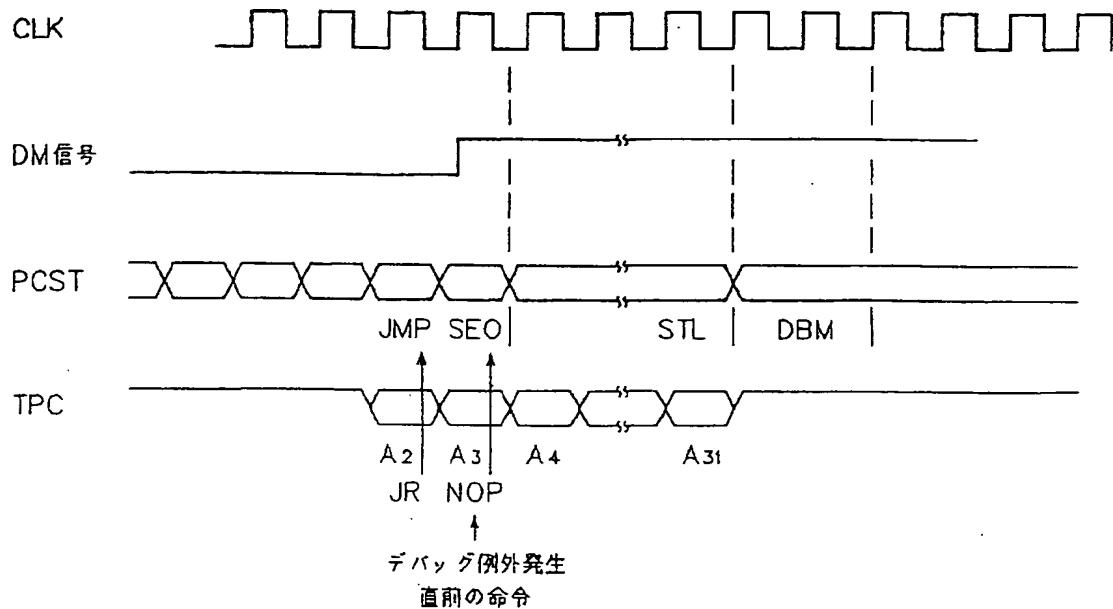
【図13】



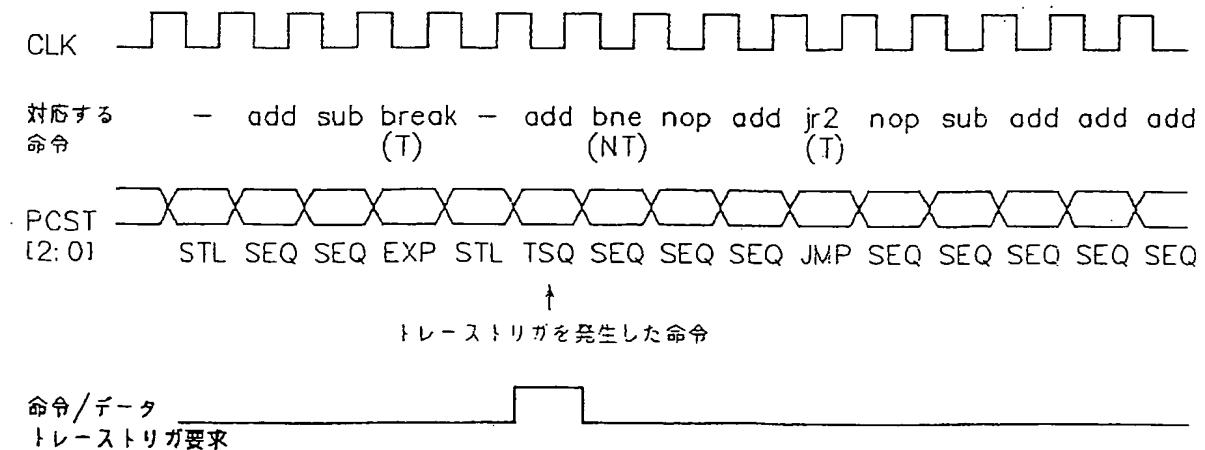
【図14】



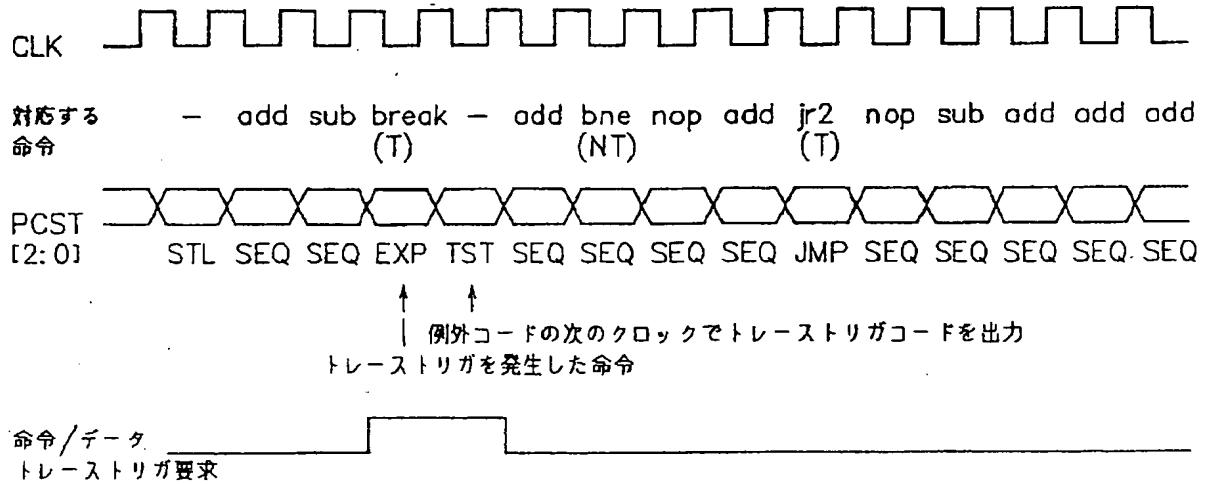
[図15]



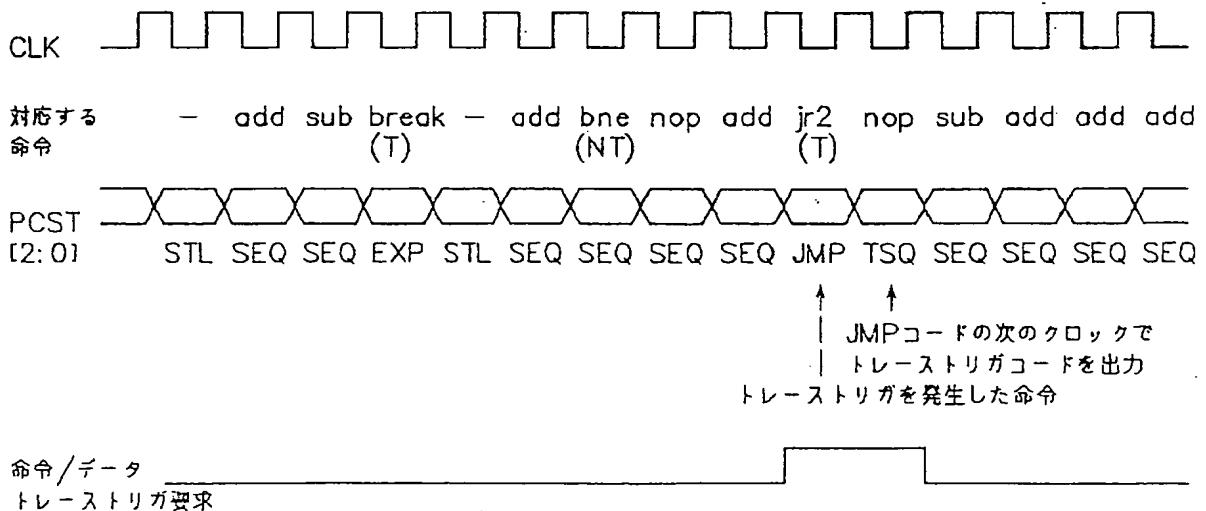
【図17】



【図18】



【図19】



【図20】

